Xerox Corporation Palo Alto Research Center

Improving Software Efficiency Through Reusable Components

Software for large, computer-driven systems was generally proprietary and highly customized in the mid-1990s. Typically, about 80 percent of each application's code was original, which meant that development and maintenance costs were high. Buyers wanted more from these programs; they wanted software that could be used in more than one application and that was easier for programmers to work with, features that would improve the programmers' productivity. Researchers at the Xerox Palo Alto Research Center (PARC) thought they could solve the problem by making software more modular, which would improve compatibility among software applications.

In 1994, PARC applied to the Advanced Technology Program's (ATP) "Component-Based Software" focused program. With ATP's cost-shared award, the two-year project began in January 1995. As researchers developed prototype applications, they began to extract parts of programming code reflecting concerns (problems that a computer program tries to solve) that cut across the entire system and to make these concerns, or aspects, separate modules. Aspects affect a programmer's choices throughout a software application, such as logging changes to data entries. This approach differed radically from earlier attempts at modularity. The PARC researchers called their approach aspect-oriented programming (AOP) and later developed products that incorporated its principles. At the same time, Java, a new programming language used to write Internet-based material, was gaining widespread popularity. AOP complemented Java because both technologies made software more modular.

When ATP funding ended in December 1997, PARC began working with the Defense Advanced Research Projects Agency to further develop AOP. PARC created a general-purpose programming language, AspectJ, which it patented. In 2002, the company made this technology freely available by transferring it to eclipse.org, an open-source project sponsored by IBM.

By 2004, AOP was well recognized in the computer industry and was taught at more than a dozen universities in North America and the United Kingdom. Eight patents emerged from the ATP-funded project. More than 3,250 articles or books have been written about AOP. In 2003, JavaWorld named AspectJ best new product or process using Java, and IBM uses AspectJ in developing new software products. The outlook for the technology is very strong.

COMPOSITE PERFORMANCE SCORE
(based on a four star rating)
* * * *

Research and data for Status Report 94-06-0036 were collected during October – December 2004.

Need for Reusability Drives Search for More Efficient Programming

In computer programming, high-level code enables programmers to use single statements to represent large chunks of binary digital code. In the mid-1990s, programmers were writing this kind of code with object-oriented programming. Object-oriented programming defined a computer program as a collection of individual units, or objects, as opposed to an earlier view in which a program was essentially a list of instructions to the computer. Each object can receive messages, process data, and send messages to other objects. Messages can be handled by one or more pieces of code, and programmers can create relationships between one object and another so that objects can inherit characteristics from other objects. For example, a system requirements engineer might specify a bank account with the associated functions of depositing, withdrawing, and providing a balance. A system designer would describe an object "bank account," which the programmer would implement in code. Object-oriented programming made it easier to create and extend complex applications such as graphical user interfaces (GUIs), operating systems, and distributed applications (for example, browsers that run on more than one computer and communicate through a network).

Programmers were also increasingly using modules, or components that could be connected to each other. Modularity enables programmers to replace or add a component without affecting the rest of the system. With component-based software, programmers could develop custom applications without concerning themselves with the details of implementation because those details were hidden. However, hiding implementation code also hid inherent problems, which meant that resolving these problems was difficult to impossible, resulting in slow and inefficient systems.

Gregor Kiczales, a computer scientist at the Xerox Palo Alto Research Center (PARC), wrote that conventional computer-industry wisdom considered it "essentially impossible for code to be both high-level and efficient, except in very specific cases that usually involve removing any hope of reusability. The programming world is thus divided into two basic camps, one that espouses elegant high-level code and the other that promotes inelegant but highly efficient code."

Xerox Corporation produces copiers, printers, image-processing systems, and desktop publishing systems that all contain computers. Xerox was interested in developing more powerful products that could be tailored to its customers' individual needs. Because that step was impossible without reusable code, the company turned to PARC. Since its founding in 1970, PARC had invented object-oriented programming and many of the technologies and products now considered standard in computing: personal computers, what-you-see-is-what-you-get (WYSIWYG) monitor display, graphical user interfaces, Ethernet networks, the mouse, and the laser printer.

Programming Languages Could Not Produce Reusable Components

There had been previous experiments in producing reusable components. For example, researchers in Japan had developed programming languages and operating systems that allowed programmers to control the behavior and location of objects, but these projects' output had not been commercialized. Researchers in France had developed control over some issues in implementing networks, but only for one highly specific use.

Separating concerns such as error-checking and handling, synchronization, and performance would improve the speed, quality, and flexibility of core code.

PARC contended that the programming then in existence could not produce software components that were compatible with other components; that is, they were not "plug and play" components. PARC hypothesized that the fundamental problem was the hidden implementation and layers of abstraction. PARC wanted to prove this through prototype applications; beyond that, it wanted to develop general principles and techniques for industry programmers. In addition, PARC wanted to achieve separation not just between functionality and implementation, but also among a number of different implementation issues.

Because the proposed prototype, principles, and techniques were radical and high risk, PARC applied for ATP funding in 1994 under the focused program, "Component-Based Software." With a larger staff made possible by the two-year award, they proposed to develop a semantic framework that describes a component's function and a separate implementation framework that prescribes how to put the component into use. The PARC team intended to articulate the principles of separation of implementation and then develop technological applications if experiments showed the principles to be sound.

Research and Business Outlooks Were Clouded

When the ATP-funded project began in January 1995, PARC faced skepticism in the research community, which predicted that the proposed solution would make code and systems less modular or flexible. There was no mechanism to open up a module to expose internal structure. It was thought that the inside of a module had to remain completely hidden for systems to be stable, robust, and flexible.

In addition, the outlook for commercialization was problematic. It was felt that the business community would be reluctant to train its programmers in a new type of programming unless there was a clear financial advantage to retraining employees, shutting down systems, and experiencing other downtime that comes with changing software. Conventional wisdom held that retraining an experienced programmer in object-oriented techniques required 6 to 18 months before that programmer was fully productive.

The PARC researchers assumed that the major problem in developing reusable components occurred when a programmer wanted to reuse some or all of the semantic code, but not necessarily all of the implementing code. As research under the ATP award began, they scaled up the project, centering their research on two customized prototypes. The prototypes included languages that application experts could use to define specific applications in specific areas: an image-understanding language for certain kinds of image-processing applications and a sparse matrix language for computations common to forms of scientific and numerical computing.

About halfway through the project, the researchers realized that separating semantic code from implementing code would not solve the problem they had set out to solve. They developed a new hypothesis based on their understanding that many of the objects were implemented in similar or duplicated code and decided to explore the difference between core concerns and cross-cutting concerns. A concern is any problem that a program tries to solve. Programs address core concerns (such as credit card billing, or sending email). Cross-cutting concerns do not relate to core concerns directly, but a program cannot be executed without addressing cross-cutting concerns. Often when programmers changed a feature of a cross-cutting concern, they had to recompile source files and check code for consistency. The team began to isolate the cross-cutting concerns, such as error-checking and handling, synchronization, and performance optimization. Separating these concerns, or aspects, the team hypothesized, would improve the speed, quality, and flexibility of core code.

Aspect-Oriented Programming Is Born

The team filed two patent applications related to sparse matrix code in 1995. Sparse matrix code enabled users to write high-level code and annotate it to make implementation more efficient. As a result, reusable components could match the speed of customized applications and were shorter and less complex.

In 1996, the PARC team sponsored a conference, Reflection ’96, to present their findings, which they called Open Implementation Analysis and Design methodology. They next presented their findings at several international conferences. At this point, the PARC researchers realized that further development would take more time and resources than could be funded by the ATP award. Therefore, although the ATP project was funded through December 1997, in 1996 the team submitted a proposal to the Defense Advanced Research Projects Agency (DARPA) for assistance to continue work after the ATP-funded project ended.

Although others had been doing research in the same general area, the PARC team was recognized as the first to articulate the technology, which they named aspect-oriented programming (AOP). Just as object-oriented programming added to the programming that preceded it, AOP added to object-oriented programming rather than replacing it.

Over the course of the ATP-funded project, interns from Northeastern University, the University of Washington, Indiana University, and Massachusetts Institute of Technology tested ways to use AOP and developed best practices and guidelines. In August 1996, the PARC team reported that they had introduced AOP to the research community and were promoting it through papers, workshops, and presentations. They filed another patent application related to a high-level language for AOP in late 1996.

Towards the end of the ATP-funded project, researchers started designing a general-purpose tool that they named AspectJ. This tool enables programmers to define features that reflect crosscutting concerns (such as security or dependability) needed for a system and, through a compiler, to place those features throughout the code. Through an alliance with PARC, the University of British Columbia systematically evaluated AspectJ in use beginning in the summer of 1997. In August 1997, PARC signed a contract with DARPA as a result of its proposal a year earlier, for a $1.3 million, three-year project. The project would explore whether AOP was suitable for use in a decentralized system so that end users could analyze and distribute information if part of the system were destroyed or if it could be used in smart-matter applications, which enable systems to adapt immediately and autonomously to changes in their environment. During this DARPA-sponsored follow-on research, PARC further developed AspectJ and readied it for dissemination.

Knowledge of AOP Spreads

The University of British Columbia evaluators and other users found AspectJ to be simple and straightforward. Components could be plugged into a software program, greatly reducing the cost and time of installation. PARC researchers were careful to create a product with a 15-minute adoption profile; that is, one that a programmer could very quickly learn alone, without further training. This profile assuaged businesses' qualms that adopting AOP would result in significant downtime.

Aspect oriented programming's modularity can help mass-merchandise web sites deliver faster, more reliable service, at a lower cost, and with greater customization.

The team benefited greatly from Sun Microsystems' introduction of Java in 1994 and its nearly instant success. Java was designed as an object-oriented programming language, using small programs downloaded from the World Wide Web to write Internet-based material that incorporated animation, sound effects, calculators, and other special effects. Java accelerated the expansion of the Internet into a flexible method of communication. AOP complemented Java, because it allowed programmers to improve productivity and quality immediately. As software becomes more complex, AspectJ allows programmers to recompose large and complex software into simpler, better targeted, higher quality products.

PARC released AspectJ to the public in 1998. Since then, "it has been available for downloading from the Internet and has been used by thousands of software developers. The only competing technology is manually installing components in a program. The costs of doing this are so prohibitive that it is not often done," according to *Benefits and Costs of ATP Investments in Component-Based Software* (NIST Report GCR 02-834).

By the end of 1999, PARC had applied for five more patents, including one for AOP. By 2001, AOP was becoming more widely known in the industry. The Association for Computing Machinery wrote that AOP "promises simpler system evolution, more comprehensible systems, adaptability, and easier reuse." AOP makes the aspect code and the target application programming easier to understand, which is especially important as software becomes more complex and as programmers are unlikely to be familiar with the intricacies of specialized algorithms.

To win broad acceptance for the new technology, PARC opted to retain the patents but make the technology open-source. In December 2002, PARC transferred AspectJ to eclipse.org, an open-source project under the sponsorship of IBM. Since then, AspectJ has become part of Java and IBM's WebSphere Suite. In June 2003, AspectJ won the *JavaWorld* Editors' Choice Award for the Most Innovative Product or Technology using Java.

According to IBM computer scientist Adrian Colyer, quality of service and serviceability were originally bundled on the Websphere platform. AspectJ allowed IBM's programmers to make quality of service and serviceability into modules that IBM can sell separately. Several of IBM's product teams incorporate AOP in their projects. In addition to developing a more modular and flexible code, some teams use aspects to enable replacement of policies (such as access or security policies) on a per-customer or per-environment basis, something that was not possible before.

In 2003, IBM started the AspectJ Development Tools (AJDT) project to provide support for developing AspectJ applications in eclipse.org's development environment. AJDT provides the tools needed to make day-to-day development with AspectJ practical for IBM development teams, as well as for developers outside IBM. AJDT has also been integrated into IBM's Rational Software Development Platform tools based on eclipse. AJDT is a separately downloaded application that can be added onto the products rather than being included in the product itself. According to Colyer, this enables both IBM and its customers to use AspectJ for enterprise application development.

IBM also uses AOP to split off the best components from its middleware (software that intermediates between two or more applications) and put them together in new ways to create more modular and flexible solutions. Typically, product teams working in this way can use common components across many environments, where previously some duplication would have been required. This ability saves several person-months of development time.

In March 2004, IBM Vice President for Software Group Strategy and Development Daniel Sabbah said, "AOP will simplify the delivery and service of high-quality software, deliver new solutions for our customers' development requirements, create opportunities for customers to add value to their software, and accelerate new initiatives at the heart of IBM's software strategy."

ATP Award Crucial to Project's Success

AspectJ is entering the mainstream of programming tools, although exact figures on usage are hard to determine because users do not have to buy the application. Six books have been published specifically about AspectJ, including Japanese and Spanish versions. AOP and AspectJ have been cited in more than 3,000 articles, papers, or presentations. PARC team leader Gregor Kiczales estimates AspectJ has between 5,000 and 10,000 users and says that AspectJ is growing as fast as it possibly can.

AOP is taught in 15 to 20 North American and United Kingdom universities. Former PARC team member Cristina Lopes continues her research in AOP as a professor at the University of California at Irvine. Xerox has patented the project's two prototypes along with AspectJ. The PARC team members have made numerous presentations, and PARC launched an AspectJ web page.

Although AspectJ was not commercialized because it is offered as a free download from the Internet, it is widely used as a tool in revenue-generating business applications. The interest in and adoption of AOP and AspectJ continues to expand both within IBM and externally, according to Colyer. Opportunities to exploit AspectJ appear and new project teams form at IBM every month.

None of this would have happened without ATP funding, according to Kiczales. Even within PARC, the idea was radical, so that receiving ATP funding was the difference between going ahead or canceling the research. In addition, PARC's relationship with ATP forced the research team to think early on how their research might lead to commercialization. Although none of the originally envisioned products were developed, ATP encouraged the team to think about the commercial problems that an advanced technology would solve, which for many software researchers was an unheard-of approach. That perspective led the team to re-target the work to the Java platform and AspectJ in the late stages of ATP funding.

As of 2004, AOP had experienced one of the fastest paths in the software industry from research to implementation; furthermore, it appeared to be slowly changing the nature of component-based software. It is recognized that computer companies can make money in the short run by selling specific components and that tools and infrastructure needed for a full-scale market take longer to reach profitability. Because of this, in Kiczales's view, the private sector often lags in developing such tools and infrastructure. ATP funding, according to Kiczales, "encourages companies to focus on the longer term technologies such as AOP that are needed for a mature, component-based software market."

"We are in the early stages of understanding the full potential of Aspect Oriented Software Development," wrote Gary Pollice, of Worcester Polytechnic Institute, in 2004. "Aspect-oriented technology might allow us to do a better job of maintaining systems as they evolve. AOP would let us add new features, in the form of concerns, to existing systems in an organized manner. The improvements in expressiveness and structure might allow us to keep systems running longer, and to incrementally improve them without incurring the expense of a complete rewrite. Using an AOP language, we might be able to test application code automatically without disturbing the code. This would eliminate a possible source of error."

Kiczales compares the benefits of AOP to those of Intel Inside. The average computer user does not understand or care about Intel's microprocessors or about aspects, but both technologies make applications work faster and more smoothly. AOP's modularity can help mass-merchandise web sites deliver faster, more reliable service, at a lower cost, and with greater customization.

Conclusion

As computer software grows more complex, programming becomes more difficult and the code produced by the programming is often unwieldy. However, the end users continue to seek higher productivity and flexibility; ultimately, they want to be able to reuse expensive software. Researchers at the Xerox Palo Alto Research Center (PARC) were looking at ways to address these concerns. They applied for and won an ATP award under the "Component-Based Software" focused program that allowed them to pursue their research on reusable software components with a larger team. In 1995, the PARC team looked at ways of separating the implementing code from the core source code. After two experiments, they modified their hypothesis to extract crosscutting concerns, or aspects, and write code to make these aspects into separate modules. This led them to formulate a new paradigm for computer software engineering, called aspect-oriented programming (AOP). In follow-on research sponsored by the Defense Advanced Research Projects Agency, the PARC team developed a highly successful product, AspectJ, which they transferred to IBM's eclipse.org, an open-source project. AspectJ is now part of the Java software platform and can be downloaded from eclipse.org. IBM is using AspectJ and AOP in its product development.

The effects of this new way of programming are just beginning to be realized. The technology has won one award and has been described in 6 books, dozens of presentations, and more than 3,000 articles. Universities in the United States, Canada, and the United Kingdom are teaching courses in AOP, and the computer industry is beginning to use it to develop new software. IBM, in particular, has expressed confidence that AOP will both simplify and improve high-quality software. The outlook for this technology is strong.

PROJECT HIGHLIGHTS
Xerox Corporation Palo Alto Research Center

Project Title: Improving Software Efficiency Through Reusable Components (Reusable Performance-Critical Software Using Separation of Implementation Issues)

Project: To develop a component software technology that separates the semantic details of a component from the implementation details in order to support the use of software components and automated software composition for high-performance applications.

Duration: 1/1/1995-12/31/1997
ATP Number: 94-06-0036

Funding (in thousands):**

ATP Final Cost	$1,670	57%
Participant Final Cost	1, 275	43%
Total	$2,945	

Accomplishments: With ATP funding, the Xerox Palo Alto Research Center (PARC) accomplished the following:

- Developed a new programming technique called aspect-oriented programming (AOP)
- Developed two prototype applications of specialized computer languages
- Developed AspectJ, an open-source language that extends Java; it is being further developed and used in IBM's software applications and by many others

In June 2003, AspectJ won the JavaWorld Editors' Choice Award for the Most Innovative Product or Technology Using Java.

PARC received the following patents for technologies resulting from the ATP project:

- "Tools for efficient sparse matrix computation"
  (No. 5,781,779: filed December 18, 1995; granted July 14, 1998)
- "Ordered sparse accumulator and its use in efficient sparse matrix computation"
  (No. 5,983,230: filed December 18, 1995; granted November 9, 1999)
- "High-level loop fusion"
  (No. 5,822,593: filed December 6, 1996; granted October 13, 1998)
- "Software constructs that facilitate partial evaluation of source code"
  (No. 6,199,201: filed August 3, 1998; granted March 6, 2001)
- "Aspect-oriented programming"
  (No. 6,467,086: filed July 20, 1999; granted October 15, 2002)
- "Aspect-oriented system monitoring and tracing"
  (No. 6,473,895: filed July 20, 1999; granted October 29, 2002)
- "Integrated development environment for aspect-oriented programming"
  (No. 6,539,390: filed July 20, 1999; granted March 25, 2003)
- "Software constructs that facilitate partial evaluation of source code"
  (No 6,631,517: filed November 2, 2000; granted October 7, 2003)

Commercialization Status: The ATP funded AspectJ is now used in a significant percentage of IBM's new products and is an open-source platform. PARC transferred AspectJ to the open-source eclipse.org project in December 2002.

Outlook: The outlook for AOP is strong. As universities include it in their curricula, more computer scientists will gain proficiency and will find ways to use it in designing programs. IBM is aware of its utility and uses it in the majority of its new software products; other software developers are likely to follow suit. AspectJ has also been designed to conform to the "15-minute rule": software engineers can download it and become productive within 15 minutes.

Composite Performance Score: * * * *

Focused Program: Component-Based Software, 1994

Company:
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Contact: Eric Steffensen
Phone: (415) 812-4073

** As of December 9, 1997, large single applicant firms are required to pay 60% of all ATP project costs. Prior to this date, single applicant firms, regardless of size, were required to pay indirect costs.

PROJECT HIGHLIGHTS
Xerox Corporation Palo Alto Research Center

Publications: Of the 6 books and more than 3,250 articles on AspectJ or AOP, the following is a sample:

- Kiczales, Gregor. “Beyond the Black Box: Open Implementation.” *IEEE Software*, 13(1), January 1996: 8-11.
- Kiczales, Gregor. “Aspect-Oriented Programming.” *ACM (Association for Computing Machinery) Computing Surveys* 28(4es), December 1996: 154.
- Coady, Yvonne, Gregor Kiczales, Michael J. Feeley, Norman C. Hutchinson, Joon Suan Ong. “Structuring operating system aspects.” *Communications of the ACM*, 44(10), October 2001: 79-82.
- Kiczales, Gregor, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. “Getting started with ASPECTJ.” *Communications of the ACM*, 44(10), October 2001: 59-65.
- Elrad, Tzilla, Mehmet Aksit, Gregor Kiczales, Karl J. Lieberherr, and Harold Ossher. “Discussing aspects of AOP.” *Communications of the ACM*, 44(10), October 2001: 33-38.
- Wand, Mitchell, Gregor Kiczales, and Christopher Dutchyn. “A semantics for advice and dynamic join points in aspect-oriented programming.” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26 (5), September 2004: 890-910.

Presentations: The following is a sample of the researchers’ presentations:

- Hannemann, Jan, and Gregor Kiczales. “Design pattern implementation in Java and AspectJ.” Proceedings of the 2002 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA 2002, November 4-8, 2002, Seattle, Washington, New York: ACM Press, 2002: 161-173.
- Kiczales, Gregor. “AspectJ: Aspect-Oriented Programming in Java.” Objects, Components, Architectures, Services, and Applications for a Networked World, International Conference NetObjectDays, NODe 2002, Erfurt, Germany, October 7-10, 2002. Eds. Mehmet Aksit, Mira Mezini, and Rainer Unland. Springer, 2003: 1.
- Devanbu, Premkumar T., Bob Balzer, Don S. Batory, Gregor Kiczales, John Launchbury, David Lorge Parnas, and Peri L. Tarr. “Modularity in the New Millennium: A Panel Summary.” Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, IEEE Computer Society, 2003: 723-725.
- Masuhara, Hidehiko, and Gregor Kiczales. “Modeling Crosscutting in Aspect-Oriented Mechanisms.” ECOOP 2003 - Object-Oriented Programming, 17th European Conference, Darmstadt, Germany, July 21-25, 2003, Proceedings. Ed. Luca Cardelli. Springer, 2003: 2-28.
- Masuhara, Hidehiko, Gregor Kiczales, and Christopher Dutchyn. “A Compilation and Optimization Model for Aspect-Oriented Programs.” Compiler Construction, 12th International Conference, CC 2003, held as part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings. Ed. Gorel Hedin. Springer, 2003: 46-60.
- Coady, Yvonne, and Gregor Kiczales. “Back to the future: a retroactive study of aspect evolution in operating system code.” Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, AOSD 2003, March 17-21, 2003, Boston, Massachusetts. New York: ACM Press, 2003: 50-59.